Information Hiding on Open Format Documents using Permutations

Michel Ruiz Tejeida and Guillermo Morales-Luna
Departamento de Computación
CINVESTAV-IPN
Mexico D.F., Mexico
Email: mruiz@computacion.cs.cinvestav.mx

***Resumen*—We introduce an information hiding protocol for textual documents arranged in open formats which are open and standards by the ISO. We show a software tool created to test the protocol's robustness on several work-group environments.**

***Index Terms*—cryptography, permutations, data hiding, digital steganography, copyright protection, ODF, PDF, factorial number system**

I. Introducción

Ocultar información en medios gráficos es una técnica que se ha utilizado por muchos años [1], dependiendo del objeto que lo oculta, se han propuesto muchas formas para incrustar y extraer secretos. La creciente actividad de crear y distribuir documentos de texto con herramientas *ofimáticas* en línea, sistemas colaborativos o por editores en dispositivos móviles, entre otros tantos ejemplos, sugiere utilizar los ficheros ODF (*OASIS Open Document Format for Office Applications*) y PDF (*Portable Document Format*) como medio anfitrión del secreto.

En la literatura podemos encontrar diferentes mecanismos empleados para ocultar información en al menos uno de los formatos mencionados.

Brassil, Low y Maxemchuk [2], describen y comparan varios mecanismos para el marcado de documentos y varios mecanismos para la decodificación de las marcas después de que los documentos han sido sometidos a tipos comunes de distorsión, con el propósito de poder identificar el documento original de alguna copia realizada ilegalmente.

Zhu, Wu y Kankanhalli [3], proponen un método de autenticación de documentos, el cual modifica el formato de descripción del documento, en donde un carácter predeterminado se permuta mediante una clave secreta del usuario. En el proceso de verificación, conociendo la clave, se determina la permutación inicial y al encontrar el carácter se puede determinar si el documento ha sido marcado. Algunas vulnerabilidades se han encontrado y corregido en este esquema, proponiendo modificar las métricas del documento para insertar una marca de agua, brindando un esquema más robusto [6].

También Artz [4], explica una forma de hacer esteganografía con permutaciones. Un documento de texto contiene objetos, que son interpretados por los programas para indicar la forma en que se debe mostrar. Para un procesador de textos los objetos le indican la forma en que se visualizarán los caracteres, como el tipo de letra, el tamaño de la letra, los títulos, y demás características. Si le damos importancia al orden en que se encuentran los objetos se puede representar como un valor oculto.

Los objetos en los formatos PDF y ODF no tienen un orden estricto, salvo algunas excepciones [7], [8]. Esto significa que un procesador de textos interpreta de la misma forma un grupo de objetos independientemente del orden en que estén escritos en el fichero, así fue posible realizar un esquema para darle un significado al orden en que se encuentren los objetos.

La organización del trabajo es la siguiente: en la Sección II se recapitulan las características y aspectos técnicos de los formatos estándares. En la Sección III se explica el protocolo de ocultamiento de información. En la Sección IV se muestran detalles de la implementación. Finalmente, se pueden encontrar las conclusiones en la Sección V.

II. Formatos estándares para documentos

Durante varios años, cuando se utilizaba una aplicación, los ficheros se almacenaban en un formato que se podía interpretar completa y adecuadamente sólo por la aplicación que los generaba. `Pages` creaba los ficheros `Pages`, `Microsoft Word` generaba los ficheros `Word`, y así sucesivamente.

Eventualmente los programadores incluían módulos para dar soporte a los formatos de sus competidores, en ocasiones podía ser un éxito, al menos hasta la siguiente actualización del formato por parte de los propietarios; lo mismo para presentaciones, hojas de cálculo y otros. El intercambio de ficheros entre diferentes aplicaciones podría ser todo un reto para los usuarios, tal así que copiar el contenido y adecuar el formato era el procedimiento realizado.

Los formatos estandarizados como el ODF y PDF llegan a solucionar el problema de almacenar e intercambiar ficheros entre aplicaciones y sistemas. Un fichero puede ser creado en `Windows` utilizando `Microsoft Word` y guardado con alguno de los formatos mencionados, al compartirlo a otro equipo con `Linux`, por ejemplo, cualquier aplicación que soporte estos formatos va a desplegar de la misma forma su contenido.

II-A. El Formato de Documento Abierto para Aplicaciones Ofimáticas

Establecido como formato estándar por la ISO (*International Organization for Standardization*) / IEC (*International*

Tabla I
CONTENIDO DE UN EMPAQUETADO ODF

Length	Method	Size	Cmpr	Name
39	Stored	39	0 %	mimetype
45	Defl:N	37	18 %	layout-cache
807121	Defl:N	76945	1 %	Pictures/18.png
10403	Defl:N	2178	79 %	content.xml
11784	Defl:N	2165	82 %	styles.xml
9187	Defl:N	1424	85 %	settings.xml
1053	Stored	1053	0 %	meta.xml
899	Defl:N	261	71 %	manifest.rdf

Electrotechnical Commission), basado en XML (*eXtensible Markup Language*) y diseñado para almacenar e intercambiar documentos entre aplicaciones *ofimáticas*, incluyendo procesadores de texto, hojas de cálculo y presentaciones, el formato ODF es neutral en cuanto a la aplicación, plataforma y sistema operativo. Se almacena como un JAR (*Java ARchive*) [5], es decir, un empaquetado zip con un fichero adicional que lista su contenido.

De acuerdo a las especificaciones técnicas [7], un fichero ODF debe cumplir ciertos requisitos, dependiendo de su estructura, puede estar constituido como un paquete (Tabla I) o como fichero único, en el primero, cada sección del formato es guardado en un fichero independiente con el propósito de facilitar su administración; este enfoque es muy parecido a la manera en que se crean sitios web, en donde se tienen ficheros para el estilo, otros para el contenido, incluso en el caso de las imágenes, estas se incorporan al empaquetado como ficheros separados que posteriormente se ligan con su ruta interna.

Cada fichero también tiene ciertas reglas, como el elemento raíz que debe tener y el tipo de contenido; sin embargo, es flexible en el orden de los atributos para cada elemento, cuidando las convenciones establecidas en el lenguaje de esquema RELAX NG (*REgular LAnguage for XML Next Generation*), el utilizado para definir el formato [9]. En la Figura 1, se muestra un extracto en donde se pueden distinguir algunos elementos que definen el estilo del contenido en documentos de texto, cada elemento tiene atributos específicos, los cuales pueden estar escritos en diferente orden sin afectar la visualización en pantalla.

```
<style:style
  style:name="P1"
  style:family="paragraph"
  style:parent-style-name="Standard"
>
    <style:paragraph-properties
      fo:margin-left="0in"
      fo:margin-right="0in"
      fo:margin-top="0in"
      fo:margin-bottom="0.1626in"
      style:contextual-spacing="false"
      style:line-height-at-least="0.1862in"
      fo:text-indent="0in"
      style:auto-text-indent="false"
      fo:padding="0in" fo:border="none"
    />
    <style:text-properties
      fo:font-variant="normal"
      fo:text-transform="none"
      fo:color="#000000"
      style:font-name="arial"
      fo:font-size="25.1000003814697pt"
      fo:letter-spacing="normal"
      fo:font-style="normal"
      fo:font-weight="normal"
    />
</style:style>
```

Figura 1. Extracto de los estilos que por defecto genera LibreOffice al crear un fichero de texto.

II-B. El Formato de Documento Portátil

El popular formato PDF es uno de fichero usado para representar documentos de manera independiente a la aplicación de software, hardware y de los sistemas operativos [8], creado y mantenido por Adobe. Un *documento* PDF consiste en una colección de objetos que juntos describen la apariencia de una o más páginas, posiblemente acompañado de otros elementos interactivos. Un *fichero* PDF contiene los objetos que lo crean junto a la información estructurada que despliega, todo se representa como una secuencia de bytes.

Las páginas de un documento, y otros elementos visuales, pueden contener cualquier combinación de texto, gráficos e imágenes. La apariencia de una página se describe a través de un *flujo de contenido*, el cual contiene una secuencia de *objetos* gráficos que serán pintados en la página.

Además, para describir el aspecto estático de las páginas, un documento PDF puede contener elementos interactivos que son posibles sólo en una representación electrónica. Se admiten anotaciones de muchos tipos de cosas, tales como texto, enlaces de hipertexto, de marcado, ficheros adjuntos, sonidos y películas. Un documento puede incluso definir su propia interfaz de usuario; acciones del teclado y del ratón, las cuales pueden ser vinculadas con acciones descritas en objetos.

Incluso, en un documento PDF puede haber información de alto nivel que es útil para el intercambio de contenidos entre aplicaciones. Es decir, puede incluir la identificación y la información de la estructura lógica que le permite realizar búsquedas, editar, o extraer información para su reutilización en otros lugares. Las contraseñas y certificados son parte de esta información de alto nivel que especifica el formato.

Un fichero PDF inicialmente se compone de cuatro elementos:

- Un *encabezado* escrito en la primer línea del PDF, identifica la versión del formato utilizado para la especificación del documento.
- El *cuerpo* de un fichero PDF consiste en una secuencia de objetos, los cuales representan el contenido del documento, como la tipografía, las páginas e imágenes. En versiones iguales o posteriores a la 1.5, el cuerpo puede contener flujos de objetos. Cada flujo de objetos contiene una secuencia de objetos indirectos.
- Una *tabla de referencias cruzadas*, localizado

```
xref
0 23
0000000000 65535 f
0000064425 00000 n
0000000019 00000 n
0000001668 00000 n
0000001689 00000 n
0000001866 00000 n
0000001906 00000 n
0000024415 00000 n
0000064568 00000 n
0000024437 00000 n
0000046596 00000 n
0000046619 00000 n
0000046819 00000 n
0000047307 00000 n
0000047648 00000 n
0000063346 00000 n
0000063369 00000 n
0000063584 00000 n
0000063985 00000 n
0000064257 00000 n
0000064300 00000 n
0000064667 00000 n
0000064764 00000 n
```

Figura 2. Ejemplo de una tabla de referencias cruzadas del formato PDF

comúnmente en la penúltima sección de la estructura estándar de un fichero, la cual muestra información que permite el acceso aleatorio de los objetos descritos en el cuerpo, de esta forma no es necesario leer todo el documento para identificar un objeto en particular. La tabla esta diseñada para tener una entrada por línea, especificando la posición del objeto dentro del documento. La tabla de referencias cruzadas, es la única parte de todo el documento que tiene un formato fijo y permite entradas de la tabla para tener acceso aleatorio. Inicialmente la tabla tiene una sola sección y conforme se forme el documento se agregan las demás entradas a la tabla. Cada sección en la tabla, como la que se presenta en la Figura 2, inicia con una línea que contiene la palabra reservada `xref`, después de esta línea se pueden agregar una o más subsecciones, sin importar el orden en que se registren. Cada subsección de referencias cruzadas contiene entradas para un rango contiguo de objetos. Cada subsección inicia con una línea que contiene dos números separados por un espacio: el primero indica el número del objeto con el que inicia la tabla, el número a su derecha representa la cantidad de entradas en la subsección. Seguida de esta línea, las consecutivas son referencias cruzadas. Cada entrada en la tabla tiene una longitud exacta de 20 bytes, incluyendo los caracteres de fin de línea. Hay dos tipos de referencias cruzadas: una para los objetos que están en uso y la otra para los que han sido eliminados y por consiguiente no se utilizan. Ambas entradas usan el mismo formato, y se pueden distinguir por la palabra clave `n` (para las que están en uso) y `f` (para las entradas libres).

- El *tráiler* de un PDF permite que la aplicación de lectura encuentre fácilmente la tabla de referencias cruzadas. Regularmente una aplicación lee un documento PDF desde el final del fichero. La última línea del fichero contiene solo una marca de fin de fichero EOF (*End-Of-File*). Antes de la marca de fin de fichero se encuentra el diccionario *tráiler*

Los *tokens* en un fichero PDF se acomodan en líneas, una línea se identifica al encontrar un carácter EOL (*End-Of-Line*), que puede ser un CR (*Carriage Return*), LF (*Line Feed*), o ambos. Los ficheros que contengan información binaria tienen líneas de longitud arbitraria; sin embargo, para mejorar la compatibilidad entre aplicaciones se recomienda no se exceda de 255 caracteres por línea.

III. Protocolo de ocultamiento

A diferencia de otras herramientas y modelos presentados, nuestro esquema no altera la información almacenada en el documento, en su lugar aprovecha las características definidas en el estándar para las aplicaciones que despliegan su contenido. En el fichero ODF es posible cambiar el orden de los atributos de una etiqueta, y para los ficheros PDF el orden de las entradas en la tabla de referencias cruzadas, como se explicó en la sección anterior. Estos n elementos son utilizados para ocultar la información, si podemos representarlos en una forma

$$F = \{f_1, f_2, \cdots, f_n\},$$

donde cada f_j sea un entero vinculado al elemento seleccionado.

Se define matemáticamente el esquema esteganográfico. Sea k una clave esteganográfica extraída del conjunto, $\mathcal{K}$, de todas las claves secretas, $\mathcal{M}$ el conjunto de todos los mensajes incrustables, $\mathcal{C}$ el conjunto de todos los objetos anfitriones y $\mathcal{W}$ el conjunto de los objetos marcados. El esquema está conformado por dos asignaciones, la de incrustación, *I*, y la de extracción, *E*:

$$I : \mathcal{C} \times \mathcal{K} \times \mathcal{M} \to \mathcal{W}$$
$$E : \mathcal{W} \times \mathcal{K} \to \mathcal{M},$$

tal que

$$\forall c \in \mathcal{C}, k \in \mathcal{K}, m \in \mathcal{M} : E(I(c,k,m),k) = m \qquad (1)$$

El conjunto de objetos anfitriones $\mathcal{C}$, esta conformado por todos los ficheros ODF y PDF. El de los mensajes incrustables está definido por

$$\mathcal{M} = \{m | m = \{0,1\}^l; 0 < l \leq \lceil \log_2[n!] \rceil\},$$

donde n es el número de elementos del objeto anfitrión seleccionado c, la función logaritmo denota la capacidad de inserción de la función *I*, su unidad son bits. El conjunto de las claves esteganográficas se denota como

$$\mathcal{K} = \{k | k \in \mathbb{N}; 0 < k < n!\},$$

donde $n!$ es el cardinal del conjunto S_n.

Se llama permutación de un conjunto a una biyección de éste sobre sí. $\forall n \in \mathbb{N}$ designaremos por S_n el conjunto de todas las permutaciones del intervalo $[1, n]$ de $\mathbb{N}$. Una forma de representar los elementos del conjunto S_n es mediante un índice N con $0 < N \leq n!$ y algunos métodos convencionales para convertir el entero en una representación habitual de una permutación, como una secuencia de la forma:

$$\pi = \{a_1, a_2, \cdots, a_n\},$$

donde cada a_i es un entero positivo menor o igual a n.

Esta representación de una permutación como un arreglo de enteros, es similar a la utilizada para almacenar los elementos del objeto anfitrión, diseñado de ésta forma para poder trabajar con los índices de las permutaciones. Una correspondencia entre n permutaciones y los primeros $n!$ números naturales asocia a cada permutacion π un índice i y se escribe $\pi = \pi_i$, se dice pues que π_i es la i-ésima permutación.

La función de incrustación *I* utiliza el mensaje a incrustar $m = \{0, 1\}^l$ codificado como un entero positivo y la clave esteganográfica k para calcular

$$x \equiv (k + m) \mod n! \tag{2}$$

después se computa la i-ésima permutación $\pi_i \in S_n$, haciendo uso de una función

$$f : \mathbb{N} \to S_n \tag{3}$$

En un proceso de mapeo se ajusta el orden de los elementos de F_0 para hacerla coincidir con el orden de los valores de π_i, a este nuevo arreglo se le llama F_1. Finalmente F_1 reemplaza a F_0 en el formato de c y como resultado tenemos el archivo marcado w.

La función de extracción *E* utiliza el objeto tentavitamente marcado w y la clave secreta k. Del archivo marcado w se localizan los n elementos permutados y se crea el arreglo $F_1 = \{f_1, f_2, \cdots, f_n\}$, como en el proceso de incrustación, y se construye la permutación $\pi_x \in S_n$, a partir de la cual se computa su índice, es decir se calcula el valor de x mediante una función

$$f^{-1} : S_n \to \mathbb{N} \tag{4}$$

con la clave k se calcula

$$m \equiv (x - k) \mod n! \tag{5}$$

y de esta forma se recupera el mensaje m.

IV. Implementación

Una herramienta de software para ocultar información en ficheros PDF y ODF siguiendo el protocolo descrito anteriormente, se implementó usando el lenguaje de programación Go, en los siguientes párrafos se describe los principales algoritmos utilizados.

Sedgewick en [10] resume varios algoritmos para generar permutaciones, e identifica el algoritmo de cambio mínimo de *Heap*, que generalmente es el más rápido para calcular todas las permutaciones de una lista con n elementos [11]. Sin embargo, para nuestros propósitos no es necesario realizar el

Require: $i \in \mathbb{N}$
Ensure: D : representación de i en base factorial
1: $b \leftarrow 1$
2: **while** $i > 0$ **do**
3: $r \equiv i \pmod b$
4: $i \leftarrow i \ \text{div} \ b$
5: $D[b] \leftarrow r$
6: $b \leftarrow b + 1$
7: **end while**
8: **return** D

Figura 3. Cambio de base decimal a factorial

Require: Índice i en el intervalo $[1..n!]$, $n \in \mathbb{N}$
Ensure: Permutación $\pi_i \in S_n$
1: $D \leftarrow$ `factorádico`(i) {Figura 3}
2: **for** $j = 1 \to n$ **do**
3: $pi_1[j] \leftarrow j$
4: $j \leftarrow j + 1$
5: **end for**
6: **for** $j = 1 \to n$ **do**
7: $a \leftarrow D[j]$
8: $\pi_i[j] \leftarrow \pi_1[a]$
9: $\pi_1 \leftarrow \pi_1 - \pi_1[a]$
10: **end for**
11: **return** π

Figura 4. Obtener i-ésima permutación

cálculo de todas las permutaciones de n, es suficiente con 1) dado un valor de índice i obtener la permutación π_i, es decir, la i-ésima permutación y 2) dada una permutación π obtener su índice i en el rango $[1..n!]$.

Como se tiene conocimiento, una forma de representar una n-permutación es mediante su índice N con $0 < N \leq n!$ y algunos métodos convencionales para convertir el entero en una representación habitual de una permutación, como una secuencia. La conversión se realiza a una secuencia de números $d_n, d_{n-1}, \cdots, d_2, d_1$, donde d_i es un entero no negativo menor que i, al arreglo $D = \{d_n, d_{n-1}, \cdots, d_2, d_1\}$ lo llamaremos representación de n en base factorial o a caso forzando el idioma factorádico.

El primer paso es, obtener la representación de N en el sistema de numeración factorial, como se describe en la Figura 3.

El segundo paso es interpretar la secuencia para generar la permutación, como se muestra en la Figura 4, se inicializa una lista en orden creciente de los elementos de S_n, $[x_1, x_2, \cdots, x_n]$, se remueve x_i, para $i = d_i$ cada uno de los elementos de la representación en el sistema de numeración factorial, y se agrega a un arreglo simple, el nuevo arreglo de elementos es la permutación π_i de n.

En el proceso contrario, dada una permutación π_i y una lista L con todos los elementos de S_n ordenada de forma ascendiente, se identifica el valor factorádico de π_i, es decir se buscan los cambios en posición de los elementos de la

Require: $\pi_i \in S_n$, $n \in \mathbb{N}$
Ensure: Índice i
1: **for** $j = 1 \rightarrow n$ **do**
2: $\pi_1[j] \leftarrow j$
3: $j \leftarrow j + 1$
4: **end for**
5: **for** $j = 1 \rightarrow n$ **do**
6: $a \leftarrow \pi_i[j]$
7: $p \leftarrow$ `EncontrarPosicion`(π_1, a) {Regresar la posición del valor de a en π_1}
8: $D[j] \leftarrow p$
9: $\pi_1 \leftarrow \pi_1 - \pi_1[p]$
10: **end for**
11: $i \leftarrow$ `Decimal`(D)
12: **return** i

Figura 5. Obtener el índice de una permutación π

permutación π_i con respecto de L, y se hace nuevamente un cambio de base del sistema de numeración factorial al decimal, este proceso está descrito en la Figura 5.

El mensaje a incrustar se lee desde la entrada estándar y se codifica a una representación entera, utilizando el código ASCII (American Standard Code for Information Interchange), como sabemos es necesario utilizar 8 bits para cada caracter y de forma similar en el proceso de extracción se realiza la decodificación del entero a su valor en la tabla ASCII. Este valor decimal correspondiente al mensaje se utiliza junto a la clave compartida para obtener el índice de la permutación.

Finalmente utilizando las funciones básicas del lenguaje Go es posible leer y analizar los ficheros ODF y PDF para poder identificar y acomodar en un arreglo los elementos permutables, al final este arreglo se modificará en relación a la permutación generada a partir de su índice. Los detalles conforme al formato seleccionado se explican en dos subsecciones.

IV-A. En documentos ODF

El documento ODF tiene la ventaja de que el autor puede establecer la cantidad de información que desea ocultar, los atributos del formato están establecidos en el manual técnico [7], con esto solamente se tendría que modificar el programa para indicar los elementos seleccionados y como se utilizarán para representar las permutaciones. En un documento de texto, por ejemplo, el empaquetado contiene un fichero exclusivo para el contenido, con nombre `content.xml`, dentro se encuentra el elemento raíz `<office:document-content>`, el cual puede tener hasta 34 atributos que se pueden asignar y permutar, lo que provee 128 bits de capacidad independientemente del contenido mismo, incluso puede ser un documento en blanco. Si se utilizan más elementos, como los estilos predeterminados del formato, o los atributos de las etiquetas correspondientes a los metadatos, la cantidad de información a ocultar puede crecer, tanto como sea posible controlar los elementos seleccionados en el programa.

IV-B. En ficheros PDF

De forma similar a un visor el programa identifica el *tráiler*, para posteriormente localizar la tabla de referencias cruzadas, que inicia con la palabra reservada `xref`, la línea inmediata indica la cantidad de elementos en la tabla, con lo que se puede crear un arreglo exacto para almacenar las entradas temporalmente, de la lista de elementos el primero debe quedar intacto para evitar problemas de compatibilidad con los programas visores, los siguientes serán utilizados para ocultar la información. A diferencia de los documentos ODF en los ficheros PDF se conoce la cantidad de información que se puede ocultar hasta realizar el análisis del fichero. Por otro lado, la cantidad de información que se puede ocultar en un documento PDF puede ser considerablemente mayor dependiendo de la cantidad de objetos utilizados para su creación, que está directamente relacionado con el tipo de contenido.

V. Conclusiones

En este trabajo, se presenta un esquema para ocultar información en los dos formatos más comúnmente utilizados para distribuir documentos de texto, ODF y PDF. Uno de los aspectos más importantes del esquema es que el fichero marcado no tiene cambios significativos, no se agregan objetos, tan solo se permutan elementos existentes, el secreto se oculta directamente en el código del formato, lo que provee un esquema de marcado invisible. La dificultad para recuperar el secreto, sin posesión de la clave, depende de la cantidad de elementos seleccionados y el tiempo que tome encontrar la permutación correcta.

El software descrito en este artículo estará disponible en http://computacion.cs.cinvestav.mx/~mruiz.

Referencias

[1] I. J. Cox, M. L. Miller, J. A. Bloom, J. Fridrich and T. Kalkel, "Digital Watermarking and Steganography", 2nd ed., San Francisco, CA, USA: *Morgan Kaufmann Publishers Inc.*, 2008

[2] J.T. Brassil, S- Low and N.F. Maxemchuk, "Copyright protection for the electronic distribution of text documents", *Proceedings of the IEEE*, vol. 87.7 pp. 1181-1196, July 1999

[3] B. Zhu, Jiankang Wu and M.S. Kankanhalli, "Render Sequence Encoding for Document Protection", *Multimedia, IEEE Transactions on* vol. 9.1, pp. 16-24, 2007

[4] D. Artz, "Digital steganography: hiding data within data", *Internet Computing, IEEE*, vol. 5.3, pp. 75-80, May 2001

[5] J David, Eisenberg, "OASIS OpenDocument Essentials", 2006

[6] M. García Horta, "Autenticación de Documentos Digitales usando la Técnica de marca de agua", tesis para obtener el grado de M. en C., Escuela Superior de Ingeniería Mecánica y Eléctrica, IPN, México, 2012.

[7] OASIS Technical Committee, "Open Document Format for Office Applications (OpenDocument) Version 1.2", version 29, September 2011

[8] Adobe Systems Incorporated, "PDF Reference, Sixth edition, version 1.23", 2006

[9] O. Uche, "Introducing OpenDocument", IBM, July 2008

[10] R. Sedgewick, "Permutation generation methods", *ACM Computing Surveys (CSUR)*, vol. 9.2, pp. 137-164, 1977

[11] S. Skiena, "Implementing discrete mathematics: combinatorics and graph theory with Mathematica", Addison-Wesley Longman Publishing Co., Inc., 1991